Hierarchic: feedback decomposition of steady-s ate equation models

Nicolas F. Rouquette
Artificial Intelligence Group
Jet Propulsion Labor atory
California Institute of Technology
M/S 525-3960
Pasadena, CA 91109
e-mail: rouquett@aig.jpl.nasa.gov
phone: (818) 306-6 I 96

Although dynamic models of differential equations are typically thought of as the "Rolls-Royce" of physical systems modeling, stcad~-state models of algebraic equations are nonetheless important for conducting various comprehensive engineering analyses of sensitivity, performance estimations and preventive diagnosis.

In this paper, we tie together notions of operationalization, ordering, and aggregation of dependencies to construct hierarchical feedback decompositions of steady-state, lumped-parameter, algebraic equation models of physical systems. This decomposition allows us to select and specify at an abstract level the nature and properties of the algorithms governing how individual equations are numerically solved. Wc illustrate an application of this technique for a two-phase ammonia thermal controller model and further show that the resulting simulation model is better, faster and cheaper than conventional simulation techniques and well-kIIowIl equation solving algorithms reviewed in the literature.

1 Simulation as model operationalization

As a mathematical modeling process [Aris, 1978], constructing simulation models of a physical system has in recent years been streamlined through automatic techniques for constructing models [Nayak, 1993; Falkenhainer and Forbus, 1992], simulation programs [Amador *et al.,* 1 993; Forbus and Falkenhainer, 1 992] and diagnostic systems [Biswas and Yu, 1993]. In this paper, we focus on constructing simulation programs for steady-state models. This problem has a misleading conceptual simplicity:

> Given the algebraic equations describing a steady-state model and numerical values for constant and input parameters, find numerical values for the model parameters that solve the model equations.

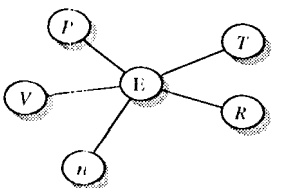
The problem stems from the lack of a good general-purpose numerical equation solving algorithm [Press *et al.,* 1 992]. Therefore, constructing a good simulation program often involves a lot of numerical analysis work to understand the mathematical properties of the equations involved and construct a corresponding equation solver. In this paper, we approach the task of constructing a simulation model as a form of operationalization, emphasizing performance, validity, and explainability:

> Given a set of algebraic equations, produce a valid, high-performance C program which computes numerical solutions to these equations.

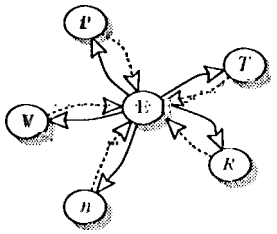
Section 2 defines our approach this problem as a new kind of ordering among parameters and equations. Next, Sec. 3 describes how to model feedback. Empirical results are summarized in Sec. 4.

2 Algebraic ordering

To efficiently reason about the possible ways to construct a simulation program for a given set of algebraic equations, we define the notion of an *algebraic* ordering *graph* that captures how each parameter of the model algebraically depends on the values of other model parameters viii the algebraic model equations. Our notion of algebraic ordering 1 wars close resenblance to the concept of causal ordering as described in Nayak's thesis [Nayak, 1993] or Levy's notion of relevance when applied to modeling [Levy, 1993]. These three notions of ordering share in common a natural representation where an equation, E: $PV = nRT$, would yield the following parameter-equation graph:



where edges indicate that an equation can causally determine a given parameter (Nayak's sense), that a parameter is relevant for the purposes of computing an equation (Levy's sense). Here, we distinguish between the fad that an equation equation can be used to compute a parameter value (shown below in solid edges) and the fact that an equation needs other parameter values to perfon m such computations (shown Mow in clashed edges)



This allows us to distinguish several ways to numerically compute parameter values. We say that an equation e can *directly* constrain the value of a parameter p when e is algebraically solvable with respect to p. Conversely, we say that an equation e *indirectly* constrains the value of a parameter p when e is not algebraically solvable with respect to p. For example, the equation: $y = \sqrt{x}$ can directly constrain y for a given x since there is a unique value of y which satisfies this equation. Conversely, this equation indirectly constrains x for a given y since there are only two possible values of x which could satisfy this equation.

Naturally, it is preferable to compute all model parameters through direct constrainment. However it, is not always possible to do so. Globally, the combinations of possible direct and indirect constrainment relationships among parameters and equations lead to five different outcomes, three for parameters and two for equations: 1) A parameter may not have any equation directly constraining it,; we say it is *under constrained* because its solution value must be guessed as there is no way to directly compute it. 2) When a parameter is directly constrained by exactly one equation, we say it is *properly constrained* because its value is unambiguously computed by solving a unique equation. 3) When multiple equations directly constrain the same parameter, we say it is *over constrained* because there is no guarantee that all such equations yield the same numerical value unless other parameters of these equations can be adjusted.

Since an equation can only be solved with respect to at most one parameter, there are only two possible outcomes. 1) An equation of n parameters which constrains a single parameter is said to be *properly constrained*: given $n-1$ known parameter values, the last one is computed by numerically or algebraically solving the equation. 2) An equation of n parameters which constrains no parameter is said to be *over constrained*: there is no guarantee that the n values found for each parameter satisfy the algebraic equation unless some of the parameter values can 1 be adjusted.

We have established a validity test for a given set, of equations and parameters which determines when a global set of indirect and direct constrainment relationships is solvable (See Ch. 4 in [Rouquette, 1995]). If all parameters and equations were properly constrainable, then a bipartite matching approach like that of Nayak's would suffice to establish a valid order of computations. To account for possible over and under constrainment, we defined an extended bipartite matching algorithm which ensures that each case of over constrainment is balanced by an adequate number of adjustable under-constrained parameters thereby resulting in a valid, computable ordering.

Like Nayak's causal ordering algorithm [Nayak, 1993], Algorithm 1 constructs a network flow graph F to match parameters and equations (Step 1 and 2). Step 3 creates paths between s and t for each exogenous parameter. The key difference with Nayak's algorithm is in the construction of paths corresponding to the possible constrainment relationships among equations and parameters: If an equation e_j can properly constrain a parameter p_i, then there is a path: $p_i \rightarrow e_j^{\text{direct}} \rightarrow e_j$ in F (Step 4), For each equation e_j, if $p_i \in P(e_j)$, then there is a path: $p_i \rightarrow e_j^{\text{indirect}} \rightarrow e_j$ to allow the possibility that e_j references p_i (Step 4). Further, since all flow paths have unit capacity, the set of all paths $p_i \rightarrow e_j^{\text{direct}} + e_j$ and $p_i \rightarrow e_j^{\text{indirect}} \rightarrow e_j$ for all p_i's and e_j's arc mutually exclusive (either e_j properly constrains p_i or e_j indirectly constrains p_i). This property confers to a maximum flow the meaning of a bipartite matching between the set of equations and parameters (step 5). Finally, the results of the matching are used to define the reference and constraint edges of the algebraic ordering: there is a reference edge from a parameter p_i to an equation e_j if $p_i \in P(e_j)$ except when e_j properly constrains p_i, (i.e., $p_i \rightarrow e_j^{\text{direct}} \rightarrow e_j$ is in the solution), in which case there is a constraint edge from e_j to p_i. It follows that when every non-exogenous parameter is properly constrained, then the algebraic ordering thus constructed is identical to a causal ordering constructed by Nayak's algorithm as long as each equation has a plausible causal interpretation. This property will help distinguish between physical and algebraic feedback loops later on.

As an example, we consider the following hypothetical set, of algebraic equations:

$$\begin{aligned}
e_1 &: f_1(P_1, P_2, P_3, P_8) = 0 \\
e_2 &: f_2(P_2, P_7) = 0 \\
e_3 &: f_3(P_3, P_4) = 0 \\
e_4 &: f_4(P_4, P_5) = 0 \\
ex_5 &: \text{exogenous}(P_5) \\
ex_6 &: \text{exogenous}(P_6) \\
e_7 &: f_7(P_4, P_6, P_7) = 0 \\
e_8 &: f_8(P_5, P_8) = 0
\end{aligned}$$

In actual circumstances with limited analytic solvability or modeler-imposed restrictions, there may be no perfect matching between equations and parameters. Such is the case for the limitations of analytic solvability of Fig. 2 which lead the extended bipartite matching algorithm to produce an actual ordering which features two under-constrained parameters, P_2 and P_4, and their corresponding over-constrained equations, e_2 and e_4.

Intuitively, the extended bipartite matching algorithm presented above combines the idea of using a perfect matching between equations and parameters as the criteria for validity but allows both direct and indirect computations. Constraint edges of the form (e, p) represent direct computations whereby the value of p is computed by e as a function of some arguments. Reference edges of the form (p, e) can be seen as indirect computations

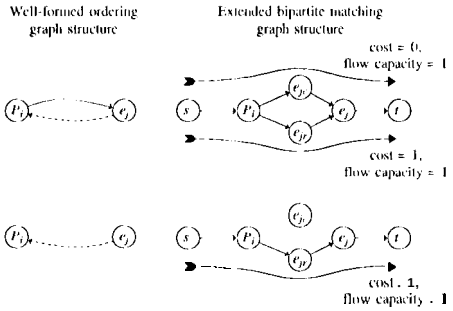
Input: A parameter-equation graph G = (V, A)
Output: A predicate: $EBM(e, p)$ defined for $e \in E$ and $p \subset P(e)$.

1) Create a network flow graph $F = (V_f, A_f)$.

2) $V_f = V \bigcup \{e^{\text{direct}}, e^{\text{indirect}} | e \in E\} \bigcup \{s, t\}$
(s and t are respectively the source and sink vertices)

3) $A_f = P_f \bigcup E_f$ where:
$P_f = \{(s, p) | p \in P\} \bigcup \{(p, ex_p), (ex_p, t) | p \in P \text{ A exogenous}(p)\}$
Each path of the form $s \rightarrow p \rightarrow ex_p \rightarrow t$
for an exogenous parameter p has unit flow capacity and zero cost.
$E_f = \{(e^{\text{indirect}}, e), (e, t) | e \in E\}$

4) The edges between each p_j and the e_i^{direct} and e_i^{indirect} vertices are defined by the edges of the parameter-equation graph according to rules below where e_{jc} stands for e_j^{direct} and e_{ji} stands for e_j^{indirect}



5) Apply a minimum cost, maximum flow algorithm on F
where the nonzero transhipment nodes are:
the source, s, with b(s) = $|P|$ and the sink, t, with $b(t) = -|P|$.

6) if $f(s,t) < |P|$ then return 0

7) Define the predicate $EBM()$ from the maximum flow topology as follows:

7a) $EBM(P_i, e_j, \text{indirect})$ holds iff $f(P_i, e_j^{\text{indirect}}) = 1$

7b) $EBM(P_i, e_j, \text{direct})$ holds iff $f(P_i, e_j^{\text{direct}}) = 1$

8) Return $EBM()$

Algorithm 1: Extended bipartite matching for constructing an algebraic ordering.

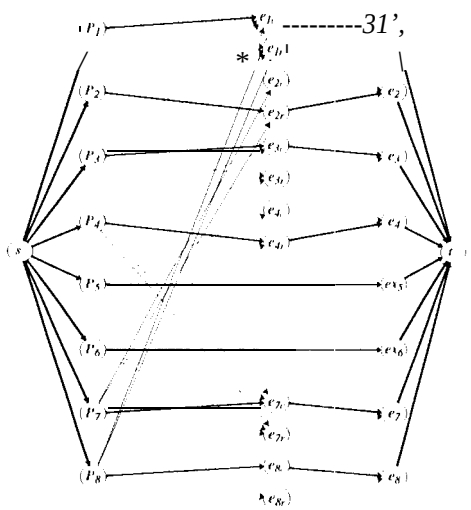


Figure 1: Example illustrating the validity of an actual ordering as an extended form of bipartite matching solvable with conventional maximum flow algorithms. The topology of the maximum flow is indicated with bold arrows. All edges have unit capacity.

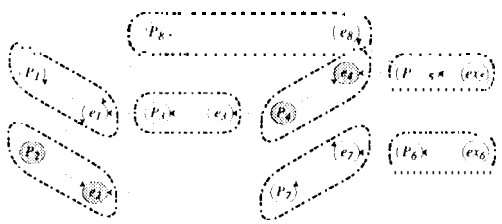


Figure ?: Algebraic ordering for the 7-equation example.

whereby the value of p is constrained by e as long as e is not directly constraining some other parameter. In terms of matchings, this observation allows us to characterize the actual valid orderings that are valid as those which have a perfect matching between the set of parameters and equations where direct and indirect forms of computation are allowed, Figure 1 illustrates this idea for the above example. The figure snows a perfect matching (bold arrows) between parameters and equation constraints and references. The perfect matching of Fig. 1 corresponds to the dashed, boxed edges of Fig. 2.

We now are faced with the task of constructing a procedure to compute all of the parameters of a model in manner consistent with the algebraic ordering found

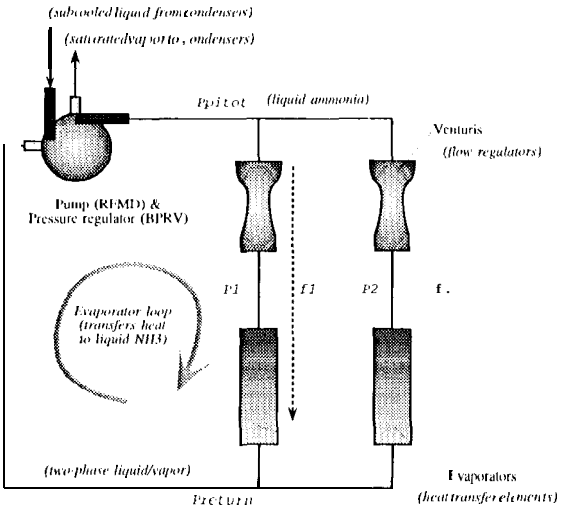


Figure 3: The evaporator loop of the Ilxtm]al-Active Thermal Control System.

for this model. If the algebraic ordering graph were acyclic, this task would be quite simple but arbitrary directed graphs can have exponentially many cycles. The next section discusses how certain cycles have a physical meaning in the notion of feedback.

3 Feedback

Feedback is a property of the interdependencies among parameters. We capture this notion in Sec. 3.1 and then proceed to analyzing feedback in Sec. 3,2.

3.1 Parameter Dependency Graph

Definition 1 (Algebraic dependency) *A parameter p' depends on p, noted by $p \rightsquigarrow p'$, iff there exists an equation e such that $p \in P(e)$ and $p' \in P(e)$ (i. e., p and p' are parameters of e) and $EBM(p, e,$ indirect) V $EBM(p', e,$ direct) holds.*

Recall from Alg. 1 that $EBM(p, e,$ direct) holds precisely for an equation $e \in E$ and a parameter $p \in P(e)$ when the value of p will be computed as a closed form expression of the remaining parameters of c. This is in contrast to $EBM(p, e,$ indirect) which holds for an equation $e \in E$ and a parameter $p \in P(e)$ when e is an algebraic constraint on the possible values that p can have.

The dependency digraph $Gd = (Pd, Ed)$ corresponding to an algebraic ordering digraph G ' = ($V = P \bigcup$ E, ,4') is defined as follows:

- $Pd = \{p \mid p \in P$ A $\neg$exogenous$(p)\}$
- $\forall p, p' \in Pd, (p, p') \in Ad$ iff $p \rightsquigarrow p'$

For notation convenience, we say that $p \rightsquigarrow^* p'$ when there exists a sequence of parameters, $p = p_1, \ldots, p_n = p'$ such that

$$p = p_1 \rightsquigarrow p_2 \cdots p_{n-1} \rightsquigarrow p_n.$$

As an illustration example, we show in Fig 3 a schematic diagram of the evaporator loop of a two-phase, External-Active Thermal Control System (EATCS) designed at McDonnell Douglas. Liquid ammonia captures heat by evaporation from hot sources (e.g., crew cabin and electronic equipment) and releases it by condensation to cold sinks (e.g. dark space). The venturis maintain a sufficiently large liquid ammonia flow to prevent complete vaporization and superheating at the evaporaters. The RFMD pump orchestrates the actual heat transfer between the two-phase evaporator return and the condenser loop (not shown).

Figure 5 shows the parameter dependency graph for one of the EATCS models presented in [Rouquette, 1995]. This model has 55 parameters, 18 of which are exogenous and the remaining 37 are to be solved with respect to 37 equations. A brute-force approach to simulating this model would consist in solving the 37 equations for the 37 unknown parameters. Ambiguities and the inherent inefficiency of this process make this approach undesirable. We seek to reformulate this brute force approach into a better, faster and cheaper simulation algorithm by abstracting each feedback loop into an efficient computational unit.

If the parameter dependency graph were acyclic, producing a simulation program would be greatly simplified: we could walk the graph in breadth-first manner and generate the simulation program from the equations thus encountered. Unfortunately, dependency graphs are often cyclic due to feedback.

3,2 Hierarchical Feedback Decomposition

Intuitively, feedback occurs when there exists at least two parameters p and p' in the dependency graph Gd such that $p \rightsquigarrow^* p'$ and $p' \rightsquigarrow^* p$ hold in Gd. Feedback is described by various terms in various scientific disciplines and engineering fields. Terms such as closed-loop circuit (as opposed to an open-loop circuit,), circular dependencies, closed-loop control, circular state dependencies, and state or control feedback are commonly used. Here, we follow some basic ideas of system theory [Padulo and Arbib, 1974] and concepts of connectedness of graph theory [Even, 1979] to distinguish two types of feedback structures, namely state and control as shown in Fig. 4.

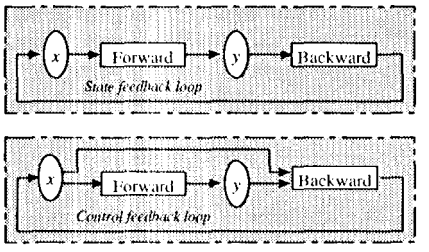


Figure 4: Connectivity of state and control feedback loops

In a state feedback loop, feedback input parameters, x, affect the feedback output parameters, y through a forward circuit,. In a dependency graph, we will have: $x \rightsquigarrow^* y$. The backward or feedback circuit in turns makes the inputs x dependent on the outputs y, or:

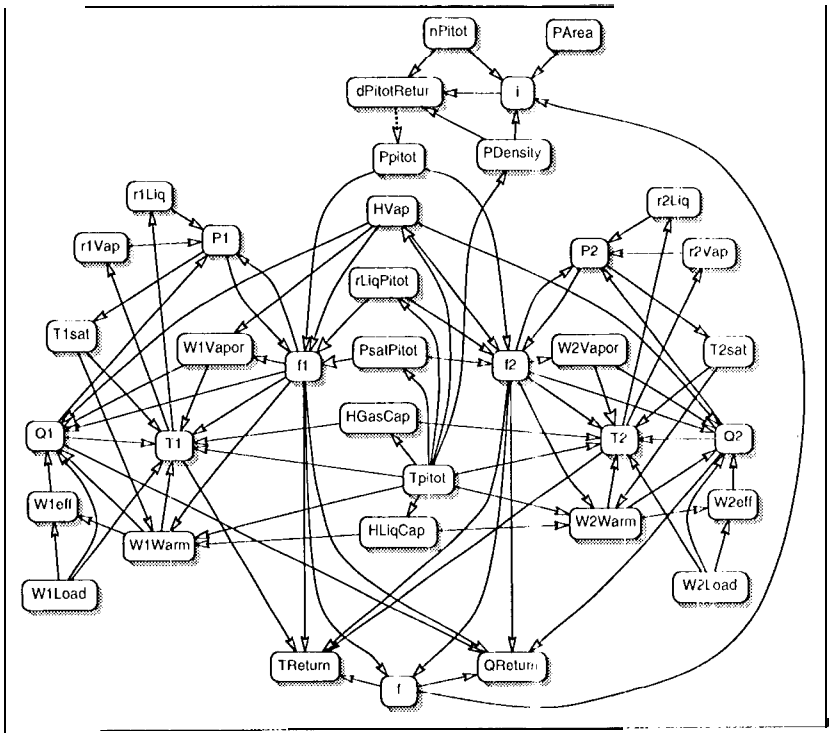


Figure 5: Dependency graph for the EATCS model.

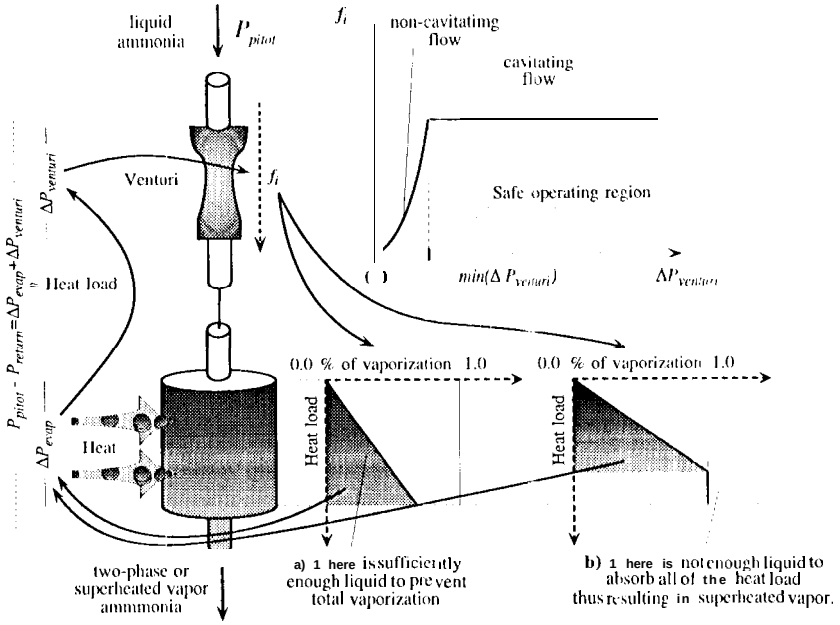


Figure 6: Acausal algebraic feedback loop: The hydrolic pressure drop, $\Delta P_{\text{venturi}}$, determines the venturi flow rate f_i (top). f_i affects the degree of valorization in the evaporator (bottom right a & b) which affects the hydrolic evaporator pressure drop ΔP_{evap}. For a fixed pitot pressure, $\Delta P_{\text{venturi}}$ will fluctuate until hydrolic balance is achieved thereby creating an algebraic 1001).

$y \rightsquigarrow^* x$. A control feedback loop is similar to a state feedback loop for the difference that the feedback circuit (usually the controller) looks at both the inputs x and the outputs y to determine the new inputs, i.e., $x, y \rightsquigarrow^* x$.

For example, the structure of the EATCS showed in Fig. 3 is a typical case of a hydraulic stale feedback loop. Feedback can also occur at the algebraic level without any physical closed circuit, involved. In the case of the EATCS, this occurs due to a combination of modeling choices resulting in circular dependencies. For example, Fig. 6 illustrates the hydro-thermal interdependencies in the EATCS venturi and evaporator legs which combine to create an algebraic feedback loop.

3.3 Operationalizing Feedback

Except for degenerate cases, a feedback loop must be solved iteratively for it corresponds to a system of $N \geq 2$ equations in N unknown parameters. Optimizing the solution quality and its computational cost requires human intervention: which equation solving technique should be used for a given feedback loop? which independent subset of parameters has optimal convergence properties? While engineers and numerical analysts arc able to address such issues, the task of deciding on the decomposition of the entire model into constituent feedback loops is comparatively more difficult. our approach is a two phase process. First, we automatically abstract the parameter dependency graph of the model equations in terms of hierarchically nested feedback loops. Then, wc present this abstract structure to the modeler who in turn specifies preferences for constructing au equation solver for each feedback loop.

Theoretically, identifying feedback in an arbitrary graph is an NP-complete problem. Fortunately, lumped-parameter algebraic models of physical systems arc typically sparse (due to lumping) and have a low degree of connectivity (because most physical components have limited interactions with neighbor components). Combined with the fact that most man-made devices are often engineered with closed-lool) control designs, it is quite common for the corresponding dependency graphs of such models to be decomposable in terms of feedback loops. Instead of designing a heuristic program of unknown capabilities, wc constructed efficient and systematic algorithms based on edge and vertex separators.

3.4 Breaking feedback loops apart,

A strongly-connected component may contain several connected sub-components interacting in complex ways. As each connected sub-component may correspond to a feedback loop, identifying the constituent feedback loops of a dependency graph therefore becomes a hierarchical graph decomposition problem.

At some level, wc need a criterion for identifying the primitive forms of feedback, namely state and control loops. Even[Even, 1979] calls a set of edges, T, an (o, b) *edge separator* if every directed path from o to b passes through at least one edge of T. Given a strongly-connected component, consider now searching for the smallest T such that, removing T makes the component unconnected or breaks it into two or more strongly-connected subcomponents. For a given pair, a, b, wc call such a subset T a one-step *optimal edge separator.*[1] Figure 7 illustrates the optimality of such separators for the purposes of breaking apart a strongly-connected component.

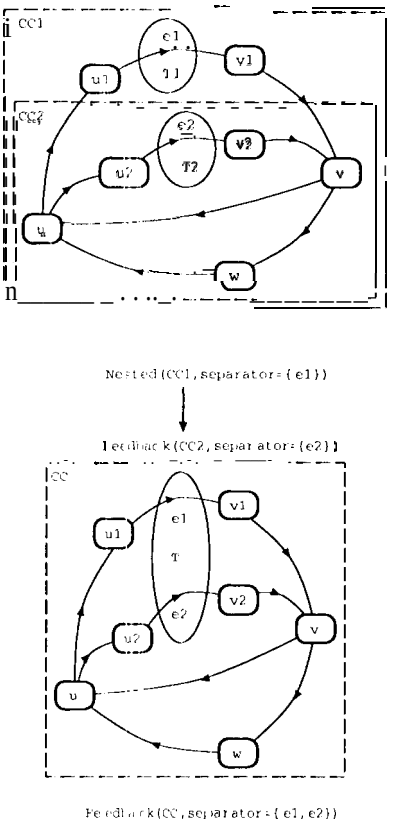


Figure 1: Minimal (top) versus one-step lookahead (bottom) edge separator and their corresponding structural decompositions. The nested decomposition is inadequate because computing v makes $v1$ and $v2$ interdependent

With a polynomial-time algorithm for finding one-step optimal edge separators (See [Rouquette, 1995, Ch. 6], wc can now describe how to hierarchically decompose strongly-connected components in terms of feedback loops.

We now have introduced sufficient machinery to present the decomposition and aggregation of dependencies (DAD) algorithm, Alg. 2. Succinctly, this algorithm solves the feedback vertex problem for algebraic dependency graphs by analyzing the effects on connectivity of removing one-step lookahead optimal edge separators. Figure 8 will help to illustrate this analysis of separability: the first column shows the effects of removing an edge separator, the middle column shows the graph

[1] One-step because we make a single analysis of how removing T affects the strong connectivity of the given subgraph.

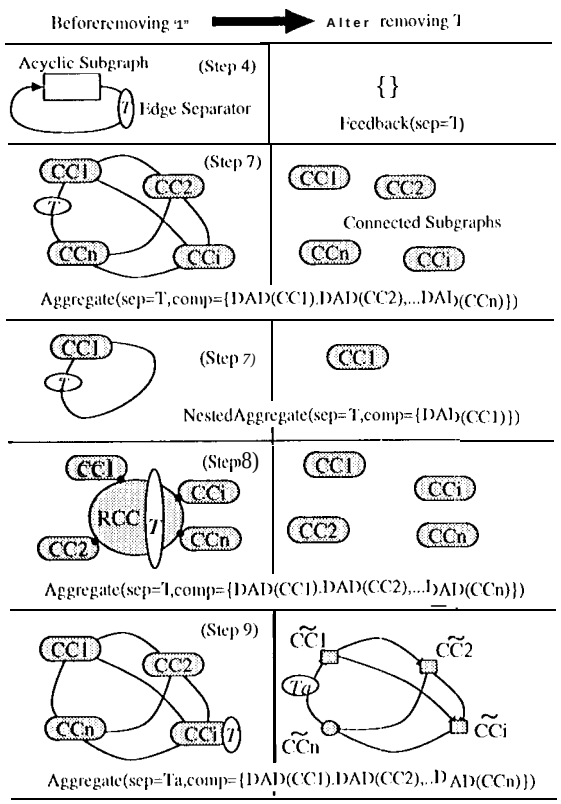


Figure 8: Possible decompositions of Alg. 2.

structure prior removal and the right column refers to the actual steps of the algorithm and the conclusions made.

Given a graph $G = (V, A)$, we analyze the effects of removing an optimal edge separator T (steps 1 and 2). G' corresponds to a simple feedback loop, Feedback(G,sep=T), when G" $= (V, A - T)$ is no longer connected (step 3 and 4). Suppose instead that removing T at step 2 did not break the connectivity of G which means that, CCS has, say, n strongly-connected sill)-coIll])ollrllt!s, $CC_1, \ldots . CC_n$. One possibility is that T is part of a set of edges which make all components Of CCS interconnected together since G it itself strongly connected. Depending on whether there is only one subcomponent or several, we have the two cases of step 7. This possibility can be easily tested by removing all edges of every component in CCS (step 5) and verifying that the resulting graph is no longer connected (step 7).

Alternatively, T can be the separator for some strongly-connected sub-component, RCC, of G which leads to two subcases. If all edges of G belong to either RCC or some other component, we have the situation of step 8 where any two sill)-(ol~~l)c)llc~lts of CCS share at most one edge[2] in such a way that) G is connected.

If there is at least one edge which dots not belong to any component of CCS or to RCC, then we have the situation of step 9 where T in fact separated a subcomponent RCC but not the whole graph. In such circumstances, DAD abstracts each sub-component as a vertex and searches for an optimal edge separator T" at the abstract level. Then, an edge separator for G independent of the sub-components is extracted from the base-lcwl edges Ta that define $\widetilde{Ta}$.

Note that the DAD algorithm is recursive since each strongly-connected sub-component of G is further decomposed. Notice that when the one:-stcl) lookahead optimal edge separator produces the empty set (step 1 and 10) or when no decomposition can be made without loosing the connectivity at the level of the sub-components (step 9), the DAD algorithm dots not decompose the graph at all. 1 n such cases, the corresponding equations of the dependency graph will have to be solved simultaneously as if they were independent algebraic constraints. If DAD manages to recursively decompose the graph into sub-components until the lowest level subcomponents are broken apart, the resulting hierarchical decomposition tree provides a computational framework to solve subsets of equations simultaneously and integrate the results of each equation solving process to obtain a global solution.

For illustration, Fig. 9 shows the 2-level feedback structure of the 01 ily strongly-connected component found in the parameter-dependency graph of the EATCS shown in Fig. 5. Since the hierarchical decompositions DA I)makes may not correspond to the modeler's perspective about feedback loops, we designed a hierarchical preference matching algorithm to guide the construction of the final equation-solving program based on the modeler's knowledge about physical and algebraic feedback.

3.5 1 Hierarchical Preference Matching

Since the DAD algorithm automatically abstracts a dependency graph into a hicarchy of feedback loops, we could stop here and use this result to construct a simulation program. 111 fact, there are compelling engineering reasons to involve the modeler: First, it helps to validate the operationalization process at the abstract level of feedback loops not only to confirm the modeler's *a-priori* expectations about what feedback loops ought, be fouud but also to dispell any possible doubts the modeler may have about the existence of a given feedback loop.

Second, feedback preferences provides the modeler with the ability to fil]c-tulle how a particular feedback loop will be simulated including the choice of the equation algorithm used, the choice of the parameters used to monitor convergence and the calculation of the initial solution estimates.

In the model decomposition of Fig. 9, the inner venturi/evaporator feedback loop (FLoop1) is, by default, structurcl with P 1 and f 1 as feedback input and output parameters. In fact, other parameters of the feedback could be used instead as long as they truly are vertex separators. In the following example, we show a matching preference where the feedback input has been replaced by the parameters: `r1l.iq` and `r1Vap`.

```
FLoop1 (id 1)
```

[2] If *two* components *cl* and *c2* shared two or more vertices, then they would be part of a larger connected component c12. Then, c12 would be part of CCS while cl and *c2* would not.

Input: $G = (V, A)$, a strongly-connected digraph
Output: The hierarchical feedback tree (111'1') decomposition of G

1) Let T be a one-step optimal edge separator of G; Return: **ComplexFeedback**(G) if $T = \emptyset$.
2) Remove T from G: $A = A - T$
3) Let $CCS = \{CC_1, \ldots, CC_n\}$ be the remaining strongly-connected components of G
4) Return: **Feedback**$(G, \mathbf{sep} = T)$ if $CCS = \emptyset$.
5) $CCS \neq \emptyset$: Remove the edges of each component in CCS and restore those of T
$A = A \bigcup T - \left(\bigcup_{cc \in CCS} A(cc)\right)$
6) Let $RCCS$ be the remaining strongly-connected components of G
7) Return: $\begin{cases} \textbf{Aggregate}(G, \mathbf{sep} = T, \mathbf{comp} = \bigcup_{cc \in CCS} DAD(cc)) & \text{if } RCCS = \emptyset \text{ and } n > 1, \\ \textbf{Nesting}(\mathbf{sep} = T, \mathbf{comp} = DAD(CC_1)) & \textbf{if } RCCS = \emptyset \text{ and } n = 1. \end{cases}$
8) (here, $RCCS$ must have only one component, RCC)
Return: **Aggregate**$(G, \mathbf{sep} = T, \mathbf{comp} = \bigcup_{cc \in CCS} DAD(cc))$ if $Ad - A(RCC) = \emptyset$.
9) (there are extraneous edges and vertices besides CCS and RCC)
Abstract the components of CCS and RCC tack into a new, unique vertex
Let $\widetilde{G} = (\widetilde{V}, \widetilde{A})$ be the resulting graph; let $\widetilde{CCS}$ be the strongly connected components of $\widetilde{G}$.
Return: **ComplexFeedback**(G) if $\widetilde{CCS} = \emptyset$.
10) Since G is strongly connected, there is only one component, i.e., $\widetilde{CCS} = \{\widetilde{CC}\}$
Let $\widetilde{Ta}$ be the optimal edge separator of $\widetilde{CC}$
Return: **ComplexFeedback**(G) if $\widetilde{Ta} = \emptyset$.
Let $Ta \subset Ad$ be the basic-level edges corresponding to $\widetilde{Ta}$.
Return: **Aggregate**$(G, \mathbf{sep} = Ta, \mathbf{comp} = (\bigcup_{cc \in CCS} DAD(cc), DAD(RCC)))$

Algorithm 2: DAI): Decomposition and Aggregation of Dependencies

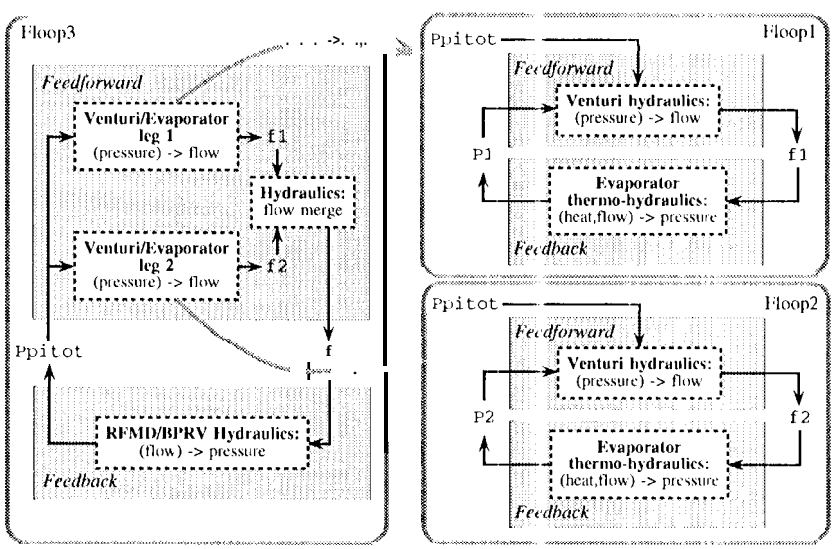


Figure 9: Physical and nested algebraic feedback loops.

```
;; specification
:structure          : state
: inputs            '(#[r1Liq]# #[r1Vap]#)
:outputs            '(#[P1]#)
```

D AGGER will use this specification as long as it matches one of the feedback loops it found. The matching criteria used consists in verifying that the the proposed inputs and outputs are vertex separators for the feedback loop.

Once the matching is successful, the remaining part of the supplied preference information is used to focus or change the way in which the actual equation-solving code will be generated.

```
FLoop1 (id 1)
  ;; operationalization
  : type                    :physical
  :input-estimators         '(#[Return +
                              W1Load/2/W1Design]#)
  : convergence-parameters  '(#[P1]# #[Q1]#)
  : convergence-threshold   0.001d0
  : convergence-method      :mathC90-dnqsol
  :max-iterations           40
  :max-transitions          1
```

where convergence-method defines the numerical algorithm to be used and the other attributes describe specific properties of that algorithm.

4 Comparison of equation solvers

For this experimental comparison, we chose the best combination of modeling assumptions for the EATCS and used the best numerical equation solving algorithm we had available [Lawsoil *et al.*, 1994]. For this comparison, we constructed three equation solvers: a brute-force program based on solving the 37 equations with respect to the 37 non-exogenous parameters; an intermediate solver, which solves the 24 equations of the unique strongly-connected component with respect to their 24 non-exogenous parameters (See Fig. 10); and a hierarchical solver, structured according to D AGGER's feedback decomposition.

Since D AGGER structures an equation solver hierarchically in terms of decomposable feedback loops, we have taken advantage of an implicit dimension reduction generating the simulation code. Each state feedback loop of the form: $y = f(x)$ and $x = g(y)$ can be restarted as a function: $x - g(f(x)) = 0$ where x is the only UNKNOWN parameter. For the EATCS decomposition shown in Fig. 9, this parameter substitution technique yields a dramatic dimension reduction as shown below:

	Number of parameters	
Loop	Before decomposition	After decomposition
FLoop1	10	1
FLoop2	10	1
FLoop3	4	1

For example, FLoop1, the feedback loop for the first venturi/evaporator leg, is reduced from 10 equations (EQ18, EQ0, EQ22, EQ23, Eq24, EQ25, EQ30, EQ32, EQ34 and EQ35) into a macro function of only one input, the estimate of PI as shown in pseudo-code below:

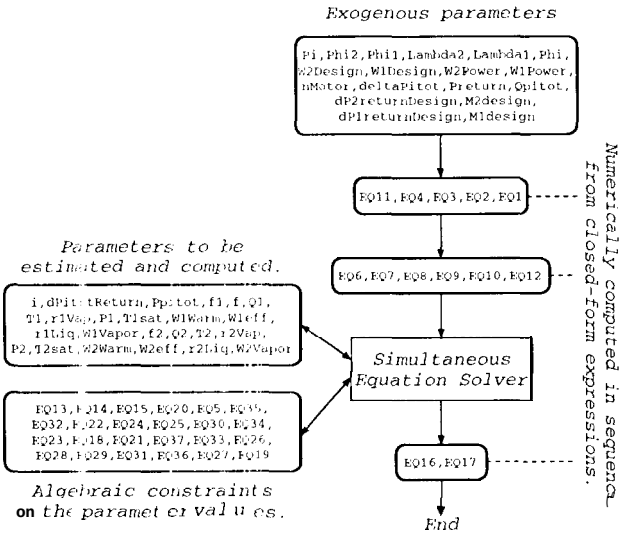


Figure 10: Structure of the intermediate equation solver

```
double residually . ..double xII, double res[] ...)
{
 PI = x[0];/* use the current estimate */

 /* from P1 calculate f1*/
 if (EQ20_test()) EQ20_true(); else EQ20_false();
 EQ18();
 EQ25 ();
 if (EQ30_test()) EQ30_true(); else EQ30_false();
 if (EQ34_test()) EQ34_true(); else EQ34_false();
 if (EQ35_test()) EQ35_true(); else EQ35_false();
 if (EQ32_test()) EQ32_true(); else EQ32_false();
 EQ23();
 EQ22();
 /* from f1, calculate the residual as:
    P1(actual) - P1(computed from f1) */
 res[0] = CEQ24();
}
```

The initial estimate of PI is computed at 10% over the return pressure, Preturn, an exogenous parameter. The other feedback loops are automatically encoded in a similar manner.

Figure 11 shows a series of plots describing a set of 20 simulation states generated from a combination of an RFMD pump slow down (Fig. 11-a), a sudden venturi clog (Fig. 11-b), and, to make matters worse, a heat load increase (Fig. 11-c). Recall that we are not simulating the dynamic response of the EATCS to these anomalous conditions. instead, we are simulating a harder problem that would occur if these anomalies would evolve slowly over a long period of time and near steady states would be observed throughout until a sudden breakdown. Each of the three solvers computed a full state prediction starting from the nominal exogenous conditions altered by nMotor, Phi1, and W1Power. The residual error of each equation solver is shown in Fig. 11-d. Although it would seem that all three solvers yield comparable solution accuracy, the brute-force and inter-

mediate solvers did not converge for the last 6 states where the initial solution estimate (37 and 24 parameters respectively!) is 100 fill' from the actual solution. For all States where convergence problems occurred, 100 equation-solving episodes were made before giving up on the solution (Fig. 11-c) while The hierarchical solver managed in al] circumstances to converge on a solution in a remarkable 6 or 7 equation-solving instances. Translated in terms of time spent, we see even stronger differences in Fig. 11-i since the 1)rutc-force solver handles 37 equations and parameters, t he intermediate solver 24, and the hierarchical solver 1 for the top-level fccd-1 back loop and 1 for each of the two lower-lcvc] feedback loops. Other differences snow up in the actual simulation results, The combination of the venturi clog, the significant pumped slow down and the significant, heat load increase caused the EATCS to overheat for a lack of liquid ammonia mass flow to transfer all of the heat. The behavior of the model in this region is extremely approximate because this is ananomalous condition; therefore, only the relative order of magnitude of the results should be used, not their actual values. Nonetheless, we note that the 1)rut(-force equation solver fails to detect the overheating condition (T1 well al bove 70 degrees) as shown in Fig. 11-f. The liquid/vapor ratios predicted by the three solvers are identical (See Fig. 11-g), while the flow rate predictions are somewhat inconsistent for the 1,lu{c-force equation solver (See Fig. 11-11).

This experiment pointed out the need of carefully choosing an initial parameter estimate for hierarchical problem solvers. Although it is true of every equation solver that the quality of the solutions obtained is limited by the quality of initial estimates, this rule applies even more to hierarchical problem solvers. Indeed, inadequate estimates can simply sidetrack the hierarchical equation solver in ways were it cannot search other alternatives. The explanation stems from the fact that a Imut,c-force equation solver has lax guidance and is therefore able to search a much larger parameter space than a hierarchical solver which has strong guidance and a much smaller parameter space to search. These differences are indicative of tradeoffs familiar in numerical analysis. In that respect, DAGGER allows the modeler to have a more intuitive understanding about what is being traded: a hierarchical structure of physical or algebraic feedback loops against the luxury of searching a large parameter space. For models of hydro-thermodynamics, it is often fruitless to search large parameter spaces because the number of independent parameters is quite small. DAGGER represents one step towards constructing a class of equation solvers where the parameter spaces being search are close to the space of independent parameters.

The experimental results confirm that using a hierarchical decomposition approach for equation-solving produces faster and better results for the EATCS. 111 other experiments with these three solvers, we noticed that, the hierarchical approach is consistently more: efficient than the other two approaches. However, the differences in solution accuracy arc more varied, sometimes Letter and sometimes worse. In this work, we used automatic techniques for recognizing and abstracting feedback loops for the task of solving algebraic equations. This resulted in faster, better, and cheaper simulation programs trading a narrow search base (fewer independent parameters) for greater computational efficiency instead of the conventional approach characterized by a computationally expansive and broad search base (all equations versus all parameters).

The research described in this paper was carried out in part by the Computer Science Department at the University of Southern California and by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

Amador *et al.*, 1993] E. Amador, A. Finkelstein, and D. Weld. Real-time self-explanatory simulation. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*. The AAAI Press, 1993.

Aris, 1978] R. Aris. *Mathematical Modeling Techniques*, volume 24 of *Research notes in mathematics*. Pitman, 1978.

[Biswas and Yu, 1993] G. Biswas and X. Yu. A formal modeling scheme for continuous systems: Focus on diagnosis. In *International Joint Conference on Artificial Intelligence*, pages 1474-1479, 1993.

[Even, 1979] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

[Falkenhainer and Forbus, 1992] B. Falkenhainer and K. Forbus. Compositional modeling of physical systems. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*. Mlrl' Press, 1992.

[Forbus and Falkenhainer, 1992] K. Forbus and B. Falkenhainer. Self-explanatory simulations: integrating qualitative and quantitative knowledge. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*. MIT Press, 1992.

[Lawson *et al.*, 1994] C. Lawson, F. Krogh, and V. Snyder. Math77 and math90, rel. 5.0: Libraries of mathematical subprograms for fortran 77 and c. Technical Report JPL D-134 f, Rev. D, Advanced Computing Applications Group, 1994.

[Levy, 1993] A. Levy. *Irrelevance Reasoning in Knowledge Based Systems*. PhD thesis, Department of Computer Science, Stanford University, 1993.

[Nayak, 1993] P. Nayak. *Automated Modeling of Physical Systems*. PhD thesis, Department of Computer Science, 1993.

[Padulo and Arbib, 1974] L. Padulo and M. A. Arbib. *System Theory*. W. B. Saunders Company, 1974

Press *et al.*, 1992] W. H. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

Rouquette, 1995] N. Rouquette. *Operationalizing Engineering Models of Steady-State Equations into Efficient Simulation Programs*. PhD thesis, Dept. of Computer Science, 1995,

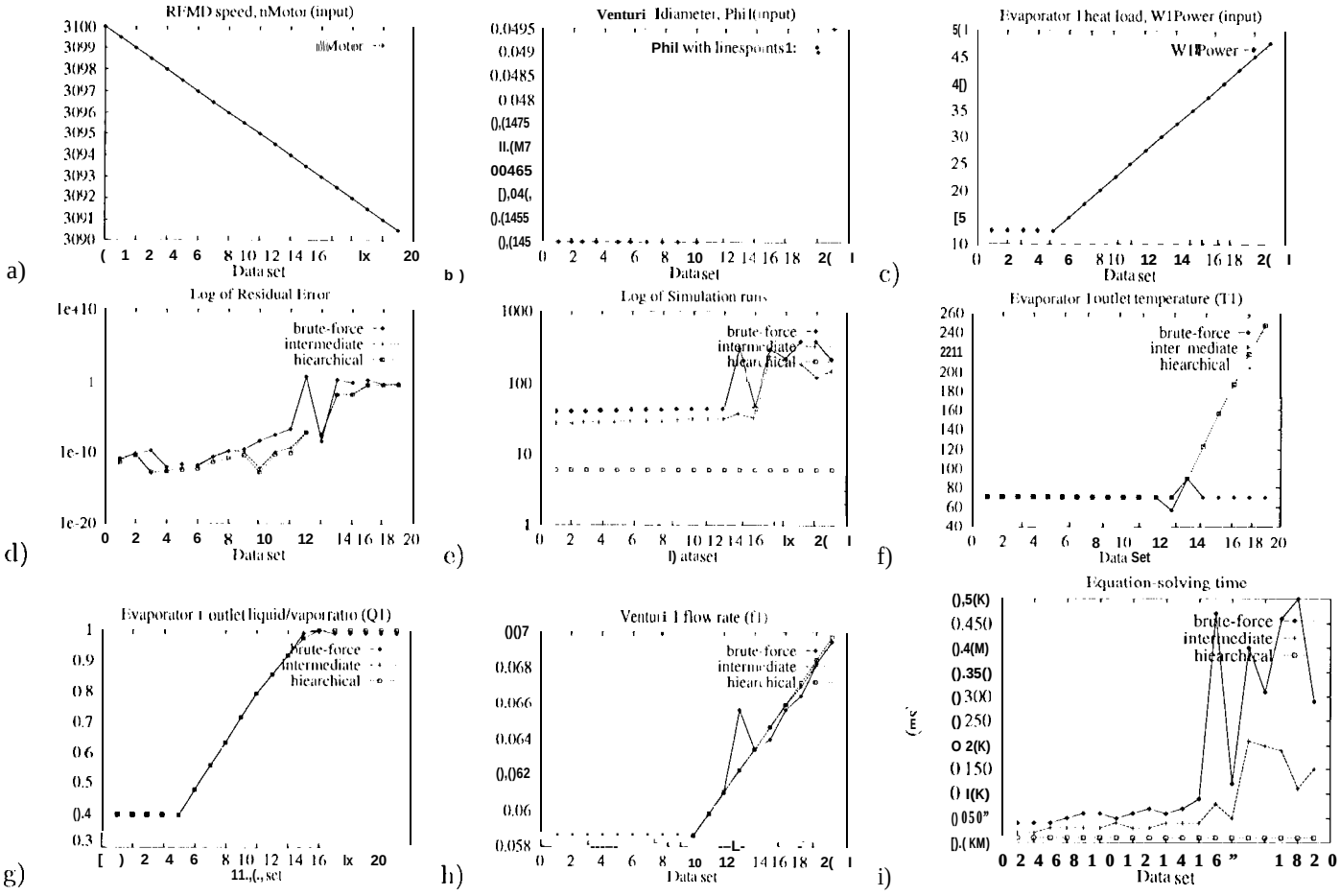


Figure 11: Empirical comparison of the three equation solvers for external input conditions (a-c). The quality of solutions found by DAGGER's hierarchical solver are generally comparable to those of the other solvers. The few exceptions where the hierarchical solver yields noticeably better solutions than the other solvers shows that given identical solution estimates, an adequately structured solver can outperform a non-structured, blind solver (a). The differences among the three solvers are most visible in terms of their computational costs (c): the hierarchical solver clearly outperforms the two others by a wide margin. As long as all solvers converge on the same solution, their behavior predictions are statistically equivalent (f-h); however, only the hierarchical solver correctly predicted the overheating (f) due to insufficient cooling power (a-b) for the given heat load (c).